Informatik Forsch. Entw. (2005) 20: 90–101
DOI 10.1007/s00450-005-0209-5

REGULÄRE BEITRÄGE

Wolfgang Reisig • Karsten Schmidt • Christian Stahl

Kommunizierende Workflow-Services modellieren und analysieren

Eingegangen am 13. Januar 2005 / Angenommen am 4. August 2005 / Online publiziert am 21. September 2005

Zusammenfassung Zur adäquaten Nutzung von Workflow-Implementierungen kommunizierender Geschäftsprozesse werden Konzepte vorgeschlagen, die von konkreten Implementierungen abstrahieren. Auf der Basis von Petrinetzen werden unterschiedliche Varianten der *Bedienbarkeit* von Workflows charakterisiert und dafür Entscheidungsalgorithmen vorgestellt. Die Angemessenheit des Ansatzes wird am Beispiel der Semantik von Komponenten der Geschäftsprozess-Modellierungssprache *BPEL* demonstriert.

Schlüsselwörter Petrinetze · offene Workflow-Netze · Workflow-Services · BPEL · Bedienbarkeit

Abstract We consider workflow implementations of communicating business processes. We propose theoretic concepts for their adequate use. Based on a class of Petri nets, we characterize different versions of *usability* (or *controllability*) of workflows and present decision procedures for these properties. Through a Petri net semantics for the web service description language BPEL, we link our concepts to a practically relevant domain.

Keywords Petri nets · Open workflow nets · Workflow services · BPEL · Controllability

CR Subject Classification I.2.2 · H.3.5 · D.2.2 · D.2.4

1 Einleitung

1.1 Kommunizierende Workflow-Services

Ein *Geschäftsprozess* ist eine funktionsübergreifende Verkettung zusammengehörender Aktivitäten, die gemeinsam einen Wert für Kunden erzeugen [21]. Unter den Aktivitäten eines Geschäftsprozesses sind unter anderem solche, die den Austausch von Informationen, Dokumenten oder realen Gütern mit Kunden, Unternehmen oder Unternehmensteilen außerhalb des Geschäftsprozesses realisieren. Diese Aktivitäten umfassen sowohl manuelle als auch mechanische Funktionen (Ausweis zeigen vs. Email senden). Im Folgenden wollen wir Aktivitäten zusammenfassend als *Kommunikationsaktivitäten* bezeichnen. Die neben dem Geschäftsprozess selbst an der Kommunikation beteiligten Akteure (Kunden, Unternehmen) nennen wir zusammenfassend *Partner*. In diesem Artikel wollen wir Geschäftsprozesse speziell unter dem Aspekt der Kommunikation mit Partnern untersuchen und dabei von anderen, ebenfalls bedeutenden Aspekten abstrahieren.

Unser Interesse am Kommunikationsaspekt von Geschäftsprozessen ergibt sich aus aktuellen Entwicklungen. Zunächst werden Geschäftsprozesse zunehmend informationstechnisch unterstützt, durch *Workflows* [1,4,17,20]. Workflows wiederum werden mehr und mehr dazu eingesetzt, die interne Abarbeitungsstruktur von *Services*, insbesondere *Web-Services* zu spezifizieren. In der aktuellen Diskussion um Architekturen verteilter, heterogener Systeme und ihrer Anwendungs-Szenarien spielen Services eine zentrale Rolle. Ein *Service* enthält ausführbare, in einer Kontrollstruktur angeordnete Operationen, eine Schnittstelle (z.B. definiert in WSDL) und einen Identifizierer (z.B. eine *URI*).

Die Kommunikation eines Service dient nicht der klassischen Form der Eingabe *vor* Beginn einer Abarbeitung und der Ausgabe *nach* Beendigung einer Abarbeitung, sondern der Kommunikation mit allen Partnern, d.h. der *Umgebung* des Service *während* einer Abarbeitung.

Die Umgebung eines Service besteht im Allgemeinen aus anderen Services, so dass in natürlicher Weise Netzwerke kommunizierender Services entstehen. Diese Architektur hat vielfältige Vorteile: Sie ist flexibel, weil sich z.B. ein neuer Service vergleichsweise einfach in ein bestehendes Netzwerk von Services integrieren oder austauschen lässt. Ein Service kann ggf. in mehreren Netzwerken verwendet werden. Vor allem aber können ggf. Services miteinander kommunizieren, die völlig unabhängig voneinander, beispielsweise bei verschiedenen Firmen, entwickelt und betrieben werden.

W. Reisig · K. Schmidt · C. Stahl
Humboldt-Universität zu Berlin, Institut für Informatik,
Tel.: +49-30-2093 3066
Fax: +49-30-2093 3067
E-mail: {reisig, kschmidt, stahl}@informatik.hu-berlin.de

Ein typisches Beispiel für einen Service ist die Organisation der Vermietung von Fahrzeugen eines Autoverleihers: Der Autoverleiher hat seine Geschäftsprozesse rechnerintegriert eingerichtet und kommuniziert entsprechend mit Kunden, Fahrzeughändlern, Versicherern etc. Damit betreibt er einen *Workflow-Service*, also einen Service, dessen Kontrollstruktur ein Workflow ist.

Beim Mieten eines Fahrzeugs realisiert auch ein Kunde einen Workflow-Service: Er weist eine Fahrerlaubnis vor, wählt eine Versicherungsvariante aus, nimmt den Autoschlüssel entgegen, etc.

Autoverleiher und Kunde schließen einen Vertrag, indem sie über eine gemeinsame Schnittstelle Informationen, Dokumente und Objekte austauschen und schließlich gemeinsam einen Endzustand erreichen.

1.2 Die Sprache BPEL

Die Kontrollstruktur eines Service kann als Software realisiert sein. Interessanterweise entwickelt sich aber gerade zur Beschreibung von Services eine Sprache, BPEL[1], zum Standard, in der Kontrollstrukturen als *Workflow*, also die operationelle Beschreibung eines Geschäftsprozesses, spezifiziert werden. BPEL wurde von IBM und Microsoft als Weiterentwicklung von WSFL bzw. XLANG im Juli 2002 vorgeschlagen. Die Sprache wird inzwischen von einem Konsortium aus über 130 Firmen unterstützt.

BPEL basiert auf bekannten Middleware-Technologien, insbesondere dem *Web Service Technology Stack*, bestehend aus *Core Layers* für den Transport und die Formatierung von Nachrichten und *Emerging Layers*, die *Quality of Service* garantieren und letztlich die verteilten Geschäftsprozesse realisieren. Teil der Emerging Layers ist auch die *Web Service Description Language* (WSDL), dem bereits etablierten Standard zur Spezifikation von Schnittstellen.

Ein BPEL-Prozess beschreibt den Aufbau eines Geschäftsprozesses innerhalb eines Web-Service und spezifiziert zugleich die Interaktion eines einzelnen Geschäftsprozesses mit den Partnern seiner Umgebung.

Der Web Service Technology Stack mit BPEL an der Spitze ist nicht der einzige Ansatz, eine Architektur für Web-Services zu definieren. Ein konkurrierender Vorschlag ist beispielsweise *Electronic Business XML* (ebXML) von OASIS. BPEL ist jedoch mit Abstand am weitesten verbreitet und anerkannt. Wir zeigen in dieser Arbeit am Beispiel der Sprache BPEL, wie sich aus in der Praxis akzeptierten Beschreibungsformen Modelle ableiten lassen, die einer theoretisch fundierten Analyse relevanter Fragestellungen zugänglich sind.

1.3 Zentrale Fragestellungen

Beim Entwurf von Workflow-Services entstehen spezifische Fragestellungen, die sich in zwei Klassen gliedern: Zum einen geht es um die Frage der genauen Bedeutung, also das *Verhalten* eines Workflow-Service. Das wird insbesondere kritisch, wenn bereits ausgeführte Aktivitäten (beispielsweise die Buchung eines Fluges) *zurückgesetzt* werden müssen (weil sich beispielsweise später herausstellt, dass kein Hotelzimmer verfügbar ist). Die zweite Klasse betrifft *Eigenschaften* von Workflow-Services, beispielsweise die Möglichkeit, einen Workflow-Service sinnvoll zu nutzen. Wir gehen auf beide Klassen ein:

Beim Verhalten von Workflow-Services unterscheidet man *positiven* Kontrollfluss, also die Formulierung des beabsichtigten zielführenden Verhaltens und den *negativen* Kontrollfluss, der das Verhalten im Fehler- und Ausnahmefall, insbesondere das Zurücksetzen (Kompensieren) von Aktivitäten betrifft. BPEL trennt beide Aspekte sehr konsequent und stellt dabei Beschreibungsmittel für die transaktionale Kapselung von Serviceteilen bereit.

Bei den Eigenschaften von Workflow-Services ist der Aspekt der *Bedienbarkeit* wichtig: Ein „vernünftiger" Workflow-Service hat einen wohldefinierten *Anfangszustand*, in dem jede Ausführung beginnt. Entsprechend hat er einen oder mehrere *Endzustände*. Die Abarbeitung eines Workflow-Service sollte nicht zu einem Deadlock führen (d.h. sowohl Workflow-Service als auch Partner warten auf Nachrichten, Dokumente oder Güter, die vom anderen aber wegen dessen Warten nicht bereitgestellt werden). Die Abarbeitung sollte außerdem nicht dazu führen, dass der Workflow-Service vom Partner gesendete Nachrichten, Dokumente oder Güter nicht verbraucht (und andersherum).

Die „Schuld" an einem Deadlock oder unverbrauchbaren Objekt kann dabei nicht immer dem Workflow-Service zugesprochen werden. Ein „böswilliger" Partner kann vernünftiges Verhalten verhindern, indem er beispielsweise Nachrichten nicht sendet, die der Workflow-Service erwartet. Andererseits ist ein Workflow-Service wenigstens insofern für „gutartige" Kommunikation verantwortlich, als seine Struktur dem Partner wenigstens die *Möglichkeit* gibt, eine einmal begonnene Ausführung eines Service bis zum Endzustand voranzubringen, also den Workflow-Service zu *bedienen*.

Wir studieren die *Bedienbarkeit* eines Workflow-Service, also die Frage nach der Existenz bedienender Partner. Dabei berücksichtigen wir, ob ein Workflow-Service mit lediglich einem Partner oder mit mehreren, unabhängig voneinander agierenden Partnern, kommuniziert. Wir stellen Lösungen für die verschiedenen Varianten der Bedienbarkeit für Workflow-Services mit zyklenfreier Kontrollstruktur vor.

1.4 Die Notwendigkeit der Modellbildung

Im letzten Abschnitt haben wir Fragen aufgeworfen, die nicht einfach zu beantworten sind. Die Schwierigkeiten beginnen schon damit, dass die Semantik von Web-Service-Beschreibungssprachen wie BPEL letztendlich bisher nur umgangssprachlich formuliert vorliegt. So sind in BPEL Feinheiten der Fehlerbehandlung, insbesondere das Zurücksetzen von Aktivitäten nicht eindeutig geklärt. Ob zwei BPEL-Prozesse

[1] *Business Process Execution Language for Web Services* [7], auch als BPEL4WS und WS-BPEL abgekürzt

P und Q *semantisch kompatibel* (d.h. P und Q können miteinander fehlerfrei interagieren), *bedienbar* (d.h. es existiert für P ein Q, so dass P und Q semantisch kompatibel sind) oder *austauschbar* (d.h. anstelle von P kann immer Q verwendet werden) sind, kann aus ihrer gegebenen syntaktischen Darstellung nicht abgeleitet werden.

Man benötigt ein *semantisches Modell* für Web-Service-Beschreibungssprachen wie z.B. BPEL. Ein solches Modell stellt die Bedeutung der Prozesse auf der gewählten Abstraktionsebene eindeutig dar und stellt zugleich Techniken bereit, die die aufgeworfenen Fragen nach Bedienbarkeit, semantischer Kompatibilität und Austauschbarkeit von Prozessen beantworten helfen.

Wir schlagen in dieser Arbeit ein solches Modell vor. Es verwendet *High-Level Petrinetze*. Die Gründe für diese Wahl werden kurz erläutert:

Workflow-Services verwenden als elementare Objekte und Operationen abstrakte Aktivitäten, beispielsweise „Antwort auf eine Nachricht“ oder „einen Auftrag stornieren“, ohne eine konkrete Repräsentation in konventionellen Datenstrukturen (Integers, Arrays). High-Level Petrinetze unterstützen das Konzept abstrakter Aktivitäten.

Für den Austausch von Nachrichten, Dokumenten und Gütern gibt es oft keine oder nur grobe Zeitvorgaben. Insbesondere können sich Nachrichten „überholen“. Die Semantik von Petrinetzen entspricht dem in natürlicher Weise.

Die Komposition einzelner Workflow-Services entspricht unmittelbar der Komposition von Petrinetzen, indem man Plätze der Petrinetz-Modelle der Prozesse verschmilzt. Aktivitäten verschiedener Prozesse wirken lokal und unabhängig voneinander, genau wie Transitionen mit disjunkten Vor- und Nachbereichen in einem Petrinetz.

Einige Aspekte von Workflow-Services, wie z.B. die Zuordnung von Rollen oder die Unterscheidung manueller von maschinellen Tätigkeiten, haben keinen wesentlichen Einfluss auf die hier studierten Fragestellungen. Sie sind daher auch nicht Bestandteil der Modellbildung.

1.5 Aufbau der Arbeit

Aus den geschilderten Sachverhalten und Fragestellungen folgt der Aufbau dieser Arbeit: Im zweiten Abschnitt wird zunächst die Semantik für Web-Service-Beschreibungssprachen am Beispiel von BPEL dargestellt. Der positive Kontrollfluss ist dabei vergleichsweise einfach zu handhaben. Seine Semantik ist auch schon an anderer Stelle beschrieben worden [10–14, 18]. Wir konzentrieren uns auf den negativen Kontrollfluss und zeigen insbesondere am Beispiel der *receive-Aktivität*, wie negativer Kontrollfluss innerhalb eines kommunizierenden Geschäftsprozesses propagiert wird. Schließlich wird noch einmal validiert, warum die gewählte Semantik angemessen ist: Mit Hilfe eines Model Checkers wurde für konkrete BPEL-Prozesse gezeigt, dass ihr Modell fundamentale semantische Eigenschaften der BPEL-Spezifikation bewahrt. Durch die Formalisierung der Semantik von BPEL können wir direkt Modelle kommunizierender Geschäftsprozesse aus realen Anwendungen ableiten.

Im dritten Abschnitt schlagen wir eine Vereinfachung der aus Workflow-Services generierten Petrinetze vor, mit dem Ziel, die computergestützte Analyse der Modelle zu vereinfachen. Weiterhin werden mit Hilfe der Petrinetz-Klasse der „offenen Workflow-Netze“ verschiedene Varianten des Begriffes der Bedienbarkeit beleuchtet und gegenüber den Soundness-Konzepten der Literatur abgegrenzt.

Im vierten Abschnitt werden Algorithmen beschrieben, um die unterschiedlichen Formen der Bedienbarkeit zu verifizieren.

2 Petrinetz-Semantik für BPEL

2.1 Zentrale Konzepte der Sprache BPEL

BPEL ist eine ausführbare Sprache zur Beschreibung kommunizierender Geschäftsprozesse. Diese Sprache wurde 2002 von IBM, Microsoft und Bea entwickelt. Nachdem das Konsortium auf über 130 Firmen anwuchs, wurde BPEL 2004 der OASIS zur Standardisierung übergeben. Der Standardisierungsprozess ist aber noch nicht abgeschlossen. Die Sprache ist in einer XML-Syntax notiert und mit englischem Fließtext informal beschrieben [7]. BPEL selbst bietet keine graphische Notation. Es existiert lediglich die *Business Process Modeling Notation* (BPMN) [28], die auch die Darstellung von BPEL unterstützt. BPEL baut auf *Aktivitäten* auf. Man unterscheidet *elementare* und *strukturierte* Aktivitäten.

Zu den elementaren Aktivitäten zählen insbesondere *invoke* (Senden einer Nachricht) und *receive* (Empfangen einer Nachricht). Eine strukturierte Aktivität umschließt eine Menge von (ggf. ebenfalls strukturierten) Aktivitäten, für die sie eine kausale Ordnung der Ausführung definiert. Beispiele sind u.a. *sequence* und *flow*, die gegebene Aktivitäten sequentiell bzw. nebenläufig anordnen. Über das Konzept des *link* können Aktivitäten eines flow kausal geordnet werden.

Eine Sonderrolle unter den strukturierten Aktivitäten nimmt der *scope* ein: Ein scope umschließt eine Aktivität für die er einen *fault handler* und einen *compensation handler* definiert. Ein fault handler erkennt Laufzeit-Fehler und korrigiert sie so weit wie möglich. Im Gegensatz dazu arbeitet der compensation handler im Stile eines Transaktionsmanagers einer Datenbank: Er macht die Effekte der vollständig abgearbeiteten Aktivität seines scope rückgängig, sofern der umgebende scope das verlangt. Ein scope und eine *Schnittstelle* beschreiben einen *BPEL-Prozess*.

Ein BPEL-Prozess kann zugleich in mehreren *Instanzen* vorliegen, die nebenläufig ausgeführt werden. Jede Instanz enthält einen Identifizierer (*correlation set*). Eine Nachricht wird einer Instanz eindeutig zugeordnet, wenn der Wert ihres correlation set sich aus dem Inhalt der Nachricht feststellen lässt. Der Wert des correlation set einer neu gebildeten Instanz wird durch die erste ein- oder ausgehende Nachricht initialisiert.

Die informelle Beschreibung von BPEL ist in einigen Aspekten ungenau, insbesondere im Bereich des compensation handling. Zudem kann die Korrektheit und andere Ei-

genschaften von BPEL-Prozessen nicht gegen eine nur umgangssprachlich gegebene Semantik verifiziert werden. Nötig ist vielmehr eine formale Semantik für BPEL.

Eine solche Semantik haben wir in [25, 26] beschrieben. Im Folgenden betrachten wir exemplarisch das semantische Modell der oben informell beschriebenen Aktivität receive in ihrem hierarchischen Kontext.

2.2 Das Muster für receive

Gegeben sei eine Instanz einer receive-Aktivität mit einer vorliegenden Nachricht im Nachrichtenkanal. Die receive-Aktivität hat zudem eine *Variable*, mit der jeweils zuletzt empfangenen Nachricht als aktuellem Wert.

In diesem Zustand verarbeitet die receive-Aktivität eine eingehende Nachricht: Entweder wird die Nachricht neuer Wert der Variable, oder die receive-Aktivität erzeugt eine Fehler-Nachricht (weil beispielsweise das correlation set der Prozess-Instanz sich nicht aus der Nachricht feststellen lässt). Während des gesamten Verarbeitungsprozesses kann die receive-Aktivität ein Stop-Signal empfangen und damit den Verarbeitungsprozess mit einer entsprechenden Nachricht an ihre Umgebung abbrechen.

Abbildung 1 zeigt die Petrinetz-Repräsentation der receive-Aktivität: Die „schwarze“ Marke auf dem Platz *initial* aktiviert die Instanz. Ihr correlation set ist c, der aktuelle Wert der Variable ist v. Zudem zeigt Abb. 1 einen Zustand, in dem eine Nachricht n den Platz *channel* erreicht hat. Damit ist t_1 aktiviert mit der Validierung $X = n$ und $CS = c$ (Durch Pfeile ohne Inschrift, beispielsweise von *initial* nach t_1, „fließt“ eine „schwarze“ Marke). Indem nun t_1 eintritt, verschwindet die „schwarze“ Marke von *initial* und die Marke n von *channel*. Auf dem Platz *running* erscheint das Paar (n, c).

Als Inschrift von t_2 und negiert als Inschrift von t_3 ist $cor(X) = CS$ ein logischer Ausdruck in den Variablen X und CS. Der Ausdruck wird zu falsch ausgewertet, wenn das mit der Funktion *cor* aus n berechnete correlation set nicht mit cs identisch ist. Außer dem eben beschriebenen Fehler können auch andere Fehler auftreten, die der Einfachheit halber in Abb. 1 nicht berücksichtigt werden. Die genaue Umsetzung in [25, 26] verallgemeinert dazu den logischen Ausdruck an den Transitionen t_2 und t_3.

Wenn der Ausdruck zu „wahr“ evaluiert wird, wird mit dem Eintritt von t_2 die Variable auf den Wert n aktualisiert und damit die Aktivitäten beendet (Marke auf *final*). Andernfalls wird über t_3 eine Marke auf den Schnittstellenplatz *failed* abgelegt. Damit haben wir den positiven Kontrollfluss von receive beschrieben: Er beginnt mit einer Marke auf *initial* und endet in *final* oder *failed*. Der positive Kontrollfluss kann jederzeit durch ein Signal auf *stop* mit einer der Transitionen t_4, t_5 oder t_7 unterbrochen werden. Einzelheiten dazu erläutert der nächste Abschn. 2.3.

Abbildung 1 zeigt das Muster für den Fall, dass das correlation set bereits initialisiert wurde. Der andere Fall, das correlation set wird durch die eingehende Nachricht initialisiert, unterscheidet sich nur in einer Kante von Abb. 1 und

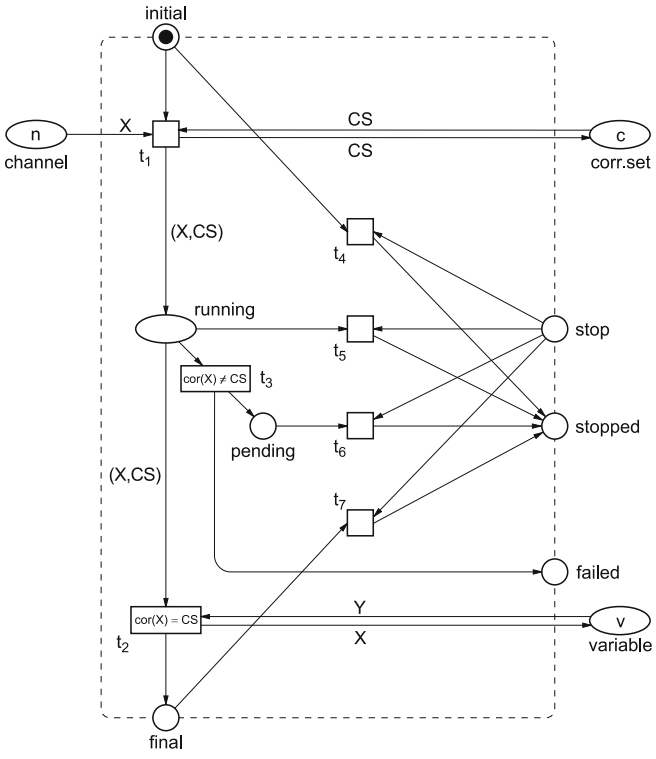

Abb. 1 receive-Muster

wird in dieser Arbeit nicht betrachtet. Wir verweisen den interessierten Leser auf [26].

2.3 Negativer Kontrollfluss

Der oben beschriebene positive Kontrollfluss des receive-Musters und aller anderen Muster ist intuitiv und formal einfach zu beschreiben. Deutlich schwieriger ist der negative Kontrollfluss: Wenn in einer Instanz einer Aktivität ein Fehler auftritt, leitet sie den entsprechenden Fehler an den fault handler ihres scope weiter. Als erstes wird der positive Kontrollfluss dieses scope gestoppt. Anschließend wird der fault handler aktiviert, der versucht, den aufgetretenen Fehler zu behandeln. Kann der fault handler den Fehler nicht behandeln, reicht er ihn an den fault handler seines umgebenden scope weiter. Kann der Fehler vom fault handler des Prozesses nicht bearbeitet werden, endet der Prozess, genauer die Prozessinstanz, fehlerhaft. Die BPEL-Spezifikation gibt lediglich die Anforderungen für das Beenden vor, z.B. ein flow wird beendet, indem alle seine eingebetteten Aktivitäten beendet werden. Die BPEL-Spezifikation macht keinerlei genaue Vorgaben, wie diese Anforderungen umgesetzt werden sollen.

In [26] schlagen wir ein Muster – das *stop-Muster* – für das Stoppen eines scope vor. Das stop-Muster stoppt die in Abb. 1 dargestellte Instanz des receive-Musters durch ein Signal auf *stop*. Mit Hilfe der Transition t_6 wird das Muster im Falle eines aufgetretenen Fehlers mit einem Signal auf *stopped* beendet. Im Gegensatz dazu verarbeiten die Transitionen t_4, t_5, t_7 das stop-Signal, indem sie den positiven Signalfluss

– wo immer er sich befindet – unterbrechen und mit einem Signal auf *stopped* beenden können. Man mag hier erwarten, dass t_4 vor t_1 und t_5 vor t_2 eine Priorität bekommt. Dies würde jedoch den asynchronen Charakter des Modells zerstören, ohne dass es die Menge der möglichen Abläufe ändert. Spätestens mit einer Marke auf *final* wird das stop-Signal über t_7 abgefangen.

Eine gesamte Prozessinstanz wird abgebrochen, indem entlang ihrer induktiven Struktur Signale an einzelne Aktivitäten propagiert werden.

2.4 Konzepte einer Petrinetz-basierten Semantik

Nachdem wir das semantische Modell der receive-Aktivität beschrieben haben, betrachten wir nun die generellen Konzepte der Petrinetz-Semantik.

Orientiert am hierarchischen Aufbau von BPEL-Prozessen beschreiben wir die Semantik von BPEL mit hierarchischen Petrinetzen. Jeder Aktivität wird ein Petrinetz, ihr *Muster*, zugeordnet. Es gibt ein zusätzliches Muster – das in Abschn. 2.3 beschriebene stop-Muster. Das Muster jeder elementaren Aktivität hat einen *initial*- und einen *final*-Platz. Über diese Schnittstelle können Muster komponiert werden. Beispielsweise ist die sequentielle Anordnung von zwei Instanzen des receive-Musters ein Muster, in dem der *final*-Platz der ersten receive-Instanz der *initial*-Platz der zweiten receive-Instanz ist. Zusätzlich müssen die beiden receive-Muster noch an die stop-Komponente des sequence-Musters „angeschlossen“ werden. Allgemein formuliert, jedes Muster einer strukturierten Aktivität hat entsprechende Steckplätze für die Muster der einzubettenden Aktivitäten. Da auch die Muster der strukturierten Aktivitäten einen *initial*- und einen *final*-Platz haben, können sie selbst wieder (wie Aktivitäten von BPEL) komponiert werden.

Die Sammlung unserer 45 Muster zusammen mit den Regeln für ihre Komposition und ihre Hierarchisierung bildet die *Petrinetz-Semantik* für BPEL [26]. Ein Muster muss die fundamentalen, semantischen Eigenschaften der entsprechenden BPEL-Aktivität bewahren, wie sie in der BPEL-Spezifikation informal beschrieben sind.

Die oben dargestellten Muster sind nicht zu verwechseln mit den Workflow-Patterns von van der Aalst et al. [2]. Workflow-Patterns stellen Anforderungen an Geschäftsprozessbeschreibungssprachen dar. Dazu gehört beispielsweise auch das Kompensieren von Aktivitäten, das in [2] informal beschrieben ist (Pattern Cancel Activity). Die BPEL-Spezifikation legt auch informal, aber wesentlich detaillierter fest, wann ein compensation handler aktiviert wird und wie dessen Ausführung aussieht. Die Petrinetz-Semantik ist operationell und implementiert die informale Beschreibung der BPEL-Spezifikation.

2.5 Validierung der Petrinetz-Semantik

Für die oben vorgestellte Petrinetz-Semantik ist es nicht möglich, formal zu beweisen, dass sie die fundamentalen semantischen Eigenschaften der informalen BPEL-Spezifikation bewahrt. Aus diesem Grund ist es notwendig, die Semantik auf ihre Plausibilität zu überprüfen. Dazu müssen wir eine hinreichend große Anzahl von BPEL-Prozessen in ein Petrinetz entsprechend der Semantik transformieren und untersuchen, ob die Muster wie erwartet zusammenspielen und ob das Modell zum entsprechenden BPEL-Prozess semantisch äquivalent ist. Zu diesem Zweck abstrahieren wir in den Petrinetz-Mustern von Daten, d.h. wir richten unser Hauptaugenmerk auf den Kontrollfluss. Wir formulieren Behauptungen über Eigenschaften des abstrakten Petrinetzmodells. Diese Behauptungen verifizieren wir mit Hilfe unseres Petrinetz-basierten Werkzeugs LoLA [22] – einem expliziten Model Checker, der leistungsstarke Reduktionstechniken implementiert.

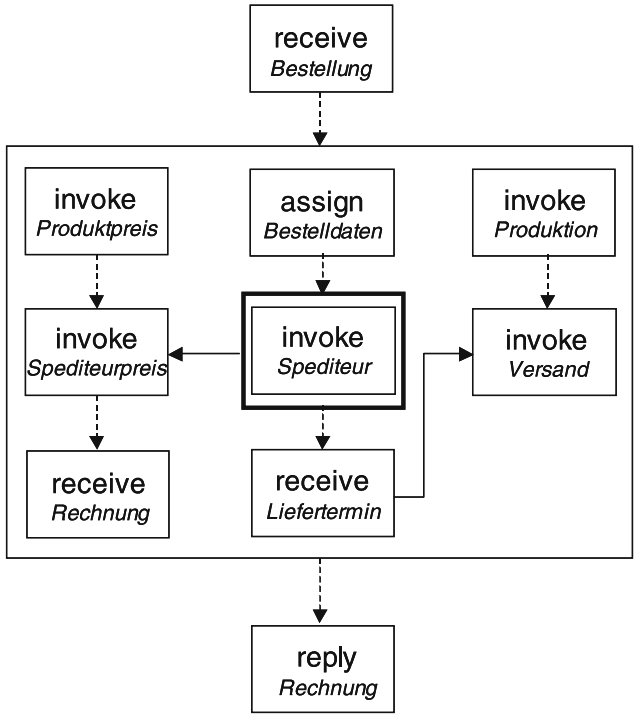

Abb. 2 Purchase Order Prozess

In [24] haben wir mit der Validierung der Petrinetz-Semantik begonnen und einen BPEL-Prozess in ein Petrinetz übersetzt. Es handelt sich dabei um eine Modifikation des Purchase Order Prozesses aus der BPEL-Spezifikation [7, S. 14 ff.], visualisiert in Abb. 2. Wir verwenden in Abb. 2 eine Variante der in Abschn. 2.1 erwähnten BPMN. Sie unterscheidet den Kontrollfluss zwischen zwei Aktivitäten von einem link zwischen den Aktivitäten.

Jede Aktivität ist durch ein Rechteck dargestellt. Einen scope oder den Prozess visualisieren wir mit einer dickeren Linie. Sequentieller Kontrollfluss ist durch gestrichelte Pfeile gekennzeichnet, Nebenläufigkeit von Aktivitäten durch die parallele Anordnung der Aktivitäten. Ein Pfeil mit durchgezogener Linie symbolisiert einen link. Betrachten wir den Ablauf: Nachdem der Prozess die Bestellung des Kunden erhalten hat (*receive*), werden drei Aufgaben parallel initi-

iert: die Berechnung des Endpreises für die Bestellung (linke Sequenz), die Auswahl eines Spediteurs (mittlere Sequenz) und die zeitliche Planung der Produktion und des Versandes (rechte Sequenz). Zwischen diesen drei Aufgaben existieren Abhängigkeiten, die mit Hilfe der links modelliert werden. Nachdem die drei Aufgaben erledigt sind, kann die Rechnung an den Kunden gesendet werden (*reply*).

Obwohl es sich bei diesem Prozess um ein einfaches Beispiel handelt, werden verschiedene Aktivitäten inklusive fault handler und links verwendet. Um die Komplexität noch etwas zu erhöhen, haben wir das *synchrone invoke* in einen scope eingebettet, so dass im Falle eines Fehlers eine Fehlernachricht gesendet werden kann. Mit Hilfe dieser Konstruktion konnten wir das Abbrechen eines eingebetteten scope untersuchen.

Das entsprechend unserer Semantik transformierte Netz hat 158 Plätze und 249 Transitionen und wurde per Hand generiert. Der Zustandsraum besteht aus 9991 Zuständen. Unter Verwendung der Reduktionstechniken konnte der Zustandsraum auf 1286 Zustände reduziert werden.

Wir haben den Prozess nach toten Plätzen und Transitionen untersucht. LoLA berechnete 15 tote Plätze und 101 tote Transitionen. Diese doch erhebliche Anzahl von Plätzen und Transitionen beruht nicht – wie man vielleicht annehmen könnte – auf einem Fehler bei der Transformation des BPEL-Prozesses, sondern ist auf die Muster zurückzuführen: Jedes Muster der Semantik deckt das vollständige Verhalten der entsprechenden Aktivität ab, insbesondere sämtliche Spezialfälle. Beispielsweise enthält der scope aus Abb. 2, der das synchrone invoke einbettet, ein stop-Muster. Das stop-Muster kann z.B. auf einen Fehler reagieren, der bei der Ausführung des compensation handler signalisiert wird. Da der scope in Abb. 2 keinen scope einbettet und somit der compensation handler bei seiner Aktivierung nichts macht, kann kein Fehler auftreten. Aus diesem Grund enthält das stop-Muster einige überflüssige Konstruktionen, die nie verwendet werden. In zukünftigen Arbeiten planen wir, die Modelle *flexibel* zur Laufzeit zu bilden. Mit anderen Worten, wir wollen die Muster erstens genau auf den jeweiligen BPEL-Prozess „maßschneidern“, und zweitens auf die Sachverhalte beschränken, die im jeweiligen Kontext benötigt werden.

Des Weiteren berechnete LoLA die 196 (erwarteten) Endzustände des Prozesses. Schließlich haben wir noch einige musterspezifische Eigenschaften verifiziert, wie beispielsweise „jeder Ablauf der *stop* enthält wird später einmal auch *stopped* enthalten“ oder „die target-Aktivität eines links tritt immer nach der source-Aktivität auf“.

Der Validierung schloss sich die Verifikation des Purchase Order Prozesses an. Hier wurden die prozessspezifischen Eigenschaften untersucht, beispielsweise die Terminierung oder das Problem, ob der Prozess dem Kunden immer eine Antwort sendet. Bei der letzten Eigenschaft handelt es sich um eine sehr komplexe temporallogische Formel, deren Gültigkeit aber von unserem Werkzeug aufgrund des kleinen Zustandsraums problemlos berechnet werden konnte.

Die Resultate bei der Validierung der Semantik mit Hilfe von LoLA zeigen die Plausibilität der Semantik. LoLA kann auch größere Prozesse verifizieren, da die Reduktionstechniken des Werkzeugs auf den aus BPEL gewonnenen Petrinetzen gut arbeiten. Diese Aussage untermauern weitere größere Fallstudien der Praxis mit einer Größenordnung von 50 Aktivitäten, die mit Hilfe von LoLA verifiziert werden konnten [16]. Die Transformation eines BPEL-Prozesses von dieser Größenordnung ermöglicht uns ein *Parser*, BPEL2PN [15], der einen in BPEL spezifizierten Prozess in ein Petrinetz entsprechend der Semantik übersetzt.

3 Abstrakte Modelle

3.1 Abstraktion

Der Hauptgrund für die Verwendung von High-Level Petrinetzen für die semantische Fundierung von BPEL ist deren Fähigkeit, Datenaspekte korrekt abzubilden. Für die Abbildung des Kontrollflusses reichen dagegen Modellierungsmittel, die bereits von *Low-Level Petrinetzen* bereitgestellt werden. Low-Level Netze haben Plätze, die ununterscheidbare (in den Abbildungen als schwarze Punkte gezeichnete) Marken tragen.

Wegen ihrer geringeren Komplexität eignen sich Low-Level Netze wesentlich besser für die Analyse relevanter Eigenschaften. Die Kontrollstruktur eines BPEL-Prozesses gewinnt man durch *Abstraktion* von Datenaspekten. Im Modell schlägt sich diese Abstraktion dadurch nieder, dass z.B. auf Kommunikationskanälen statt des Inhalts einer Nachricht nur noch die Anwesenheit einer Nachricht modelliert wird. Wenn inhaltliche Aspekte von Nachrichten den Kontrollfluss beeinflussen, kann man einen Kanal für solche Nachrichten durch mehrere Kanäle ersetzen. Hat z.B. im High-Level Modell eine Nachricht einen Wahrheitswert als Inhalt, führt man im Low-Level Modell zwei Kanäle ein, wobei die Anwesenheit einer Nachricht in einem der Kanäle eine Nachricht mit Inhalt *true* und eine Nachricht im anderen Kanal eine Nachricht mit Inhalt *false* repräsentiert.

Mit ähnlichen Konstruktionen kann man auch andere besonders wichtige Datenabhängigkeiten mit den Mitteln von Low-Level Petrinetzen ausdrücken. Einen geeigneten Grad an Abstraktion kann man entweder manuell, auf der Grundlage von Kenntnissen über das modellierte System, oder durch einen automatischen Prozess der schrittweisen Abstraktionsverfeinerung [5] gewinnen.

3.2 Offene Workflow-Netze

Die Abstraktion von BPEL-Prozessen führt zu Petrinetzen mit einigen typischen strukturellen Merkmalen. Auf der Basis dieser Merkmale wird in diesem Abschnitt eine Klasse von Petrinetzen, *offene Workflow-Netze (oWFN)*, vorgestellt und in den folgenden Abschnitten weiter studiert. Dabei ist es im weiteren Verlauf dieses Artikels nicht mehr relevant, ob die untersuchten Netze aus BPEL-Prozessen entstanden oder anderweitig generiert wurden.

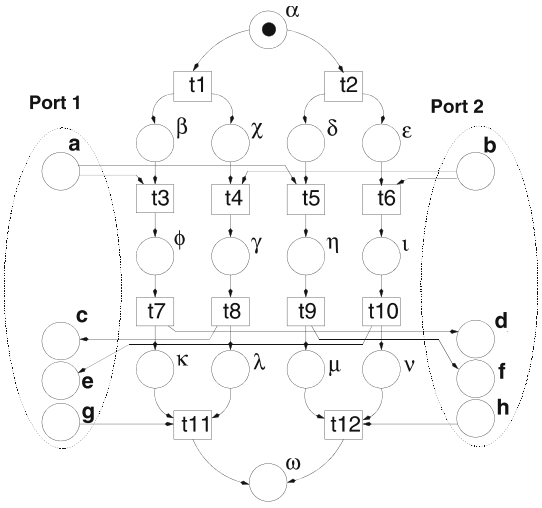

Abb. 3 oWFN. Die umrandeten Plätze sind Schnittstellenplätze. **a**, **b**, **g** und **h** sind Eingangskanäle, **c**, **d**, **e** und **f** Ausgangskanäle. Die beiden Umrandungen sollen andeuten, dass für dieses Netz zwei unabhängig agierende Partner vorgesehen sind

Abbildung 3 zeigt ein oWFN. In einem oWFN gibt es *Anfangs-* und möglicherweise mehrere *Endmarkierungen*. In Abb. 3 ist die Anfangsmarkierung dargestellt. Das Netz hat eine einzige Endmarkierung: ω trägt genau eine Marke, alle anderen Stellen sind unmarkiert. Weiterhin gibt es in einem oWFN *Schnittstellenplätze* (in Abb. 3 die Stellen **a** bis **h**). Ein Schnittstellenplatz modelliert einen asynchronen Nachrichtenkanal zu einer nicht modellierten Umgebung. Die Schnittstellenplätze sind unterschieden in *Eingangs-* und *Ausgangskanäle*. Ein Eingangskanal (in Abb. 3 **a**, **b**, **g** und **h**) hat keine Transitionen in seinen Vorbereich, ein Ausgangskanal (die übrigen Schnittstellenplätze in Abb. 3) keine Transitionen in seinem Nachbereich. Die Schnittstellenplätze sind partitioniert in *Ports*. In Abb. 3 gibt es zwei Ports, {**a**, **c**, **e**, **g**} und {**b**, **d**, **f**, **h**}. Die Aufteilung in Ports repräsentiert die Tatsache, dass die Umgebung eines oWFN aus unabhängigen, autonom agierenden Partnern besteht. So hat ein Online-Reisebüro z.B. Schnittstellen zu einem Kunden, zu einem Flugreservierungssystem und einem Hotelreservierungssystem. Ein oWFN-Modell eines solchen Reisebüros würde dann die Schnittstellenplätze in drei Ports partitionieren.

Die Markierung der Schnittstellenplätze kann nicht nur im oWFN selbst, sondern auch von der Umgebung des Prozesses verändert werden. Eine Umgebung kann Marken von Ausgangskanälen entfernen und Marken auf Eingangskanälen erzeugen.

Offene Workflow-Netze umfassen echt die Klasse der *Workflow-Netze* [1]. Der entscheidende Unterschied sind dabei die in [1] nicht vorgesehenen Schnittstellenplätze. Wegen der dadurch modellierten Offenheit gegenüber einer Umgebung ergeben sich völlig neuartige Fragestellungen. Eine dieser Fragestellungen, *Bedienbarkeit*, wird im nächsten Abschnitt vorgestellt.

3.3 Bedienbarkeit

In [1] wird für Workflow-Netze der Begriff der *Soundness* studiert. Ein Workflow-Netz ist sound, falls es von jeder erreichbaren Markierung aus in einen Endzustand übergehen (terminieren) kann und jede Transition aktivierbar ist. Der hier vorgestellte Begriff der Bedienbarkeit passt den Begriff der Soundness auf die Klasse der oWFN an.

Ob ein oWFN terminiert oder in einen Deadlock gerät, hängt im allgemeinen davon ab, ob die Umgebung „protokollgerecht“ agiert. Terminierung bei *jedem* möglichen Verhalten der Umgebung zu verlangen, wäre sicher ein zu starkes Kriterium, da eine böswillige Umgebung immer in der Lage ist, erwartete Nachrichten zu verweigern oder unerwartete Nachrichten zu senden. Wir können aber ganz sicher ein oWFN N dann als fehlkonstruiert ausweisen, wenn es für die Umgebung überhaupt keine Möglichkeit gibt, so mit N zu interagieren, dass N terminiert.

Wir können uns das Verhalten einer bestimmten Umgebung, ebenfalls durch ein oWFN modelliert, vorstellen. Ein solches oWFN wollen wir *Partner* nennen, wobei wir, um der im vorigen Abschnitt motivierten Strukturierung der Umgebung in unabhängige Teile gerecht zu werden, je einen Partner pro Port vorsehen. Ein *Partner* für einen Port π eines oWFN N ist ein oWFN M mit einem einzigen Port, der genau die Stellen in π umfasst, aber in umgekehrter Ausrichtung: Ist eine Stelle aus π Ausgangskanal in N, so ist sie Eingangskanal ist M und umgekehrt. Abgesehen von den Schnittstellenplätzen, setzen wir die Elemente von N und M als disjunkt voraus.

Für ein oWFN und seine Partner ist der Begriff der *Komposition* unmittelbar einsichtig: Es werden sämtliche Stellen, Transitionen und Kanten vereinigt. Die Anfangsmarkierung einer Stelle ergibt sich als die Summe aller Marken auf dieser Stelle in denjenigen Netzen, in denen sie vorkommt. Endmarkierungen werden analog gebildet, für jede Kombination aus je einer Endmarkierung pro beteiligtem oWFN. Im komponierten System gibt es keine Schnittstellenplätze mehr.

Auf der Basis der Begriffe *Partner* und *Komposition* definieren wir nun den Begriff Bedienbarkeit: Ein oWFN N heißt *bedienbar*, falls Partner (einer pro Port von N) derart existieren, dass im komponierten Netz von jeder Markierung aus eine Endmarkierung erreichbar ist. Ist dabei jede Transition von N im komponierten System aus N und wenigstens einem Tupel von Partnern aktivierbar, so nennen wir N *vollständig bedienbar*. Da das komponierte System keine Schnittstellenplätze enthält, also im wesentlichen ein Workflow-Netz nach [1] ist[2], heißt vollständige Bedienbarkeit also die Existenz von Partnern, mit denen komponiert N sound ist.

Von einer Menge von Partnern mit den gesuchten Eigenschaften sagen wir, sie *bedienen* N bzw. bedienen N *vollständig*.

[2] es gibt bei van der Aalst einige syntaktische Einschränkungen, z.B. die Existenz je eines Start- und Endplatzes, auf die wir bei oWFN verzichten.

In [8] wird der Begriff der *relaxed Soundness* als die Existenz eines Schedulers für ein Workflow-Netz N definiert, der das Verhalten von N so beschränkt, dass N sound wird. Der dort verwendete Begriff des Schedulers unterscheidet sich signifikant von unserem Begriff von Partnern. Während unsere Partner über die Schnittstellen asynchron und „von außen“ mit N wechselwirken, agiert der Scheduler aus [8] synchron und im „Inneren“ von N, hat insbesondere direkten Zugriff auf die Markierung sämtlicher Stellen von N (nicht nur der Schnittstellenplätze).

Der Begriff der Bedienbarkeit stellt daher ein neues Konzept dar und verlangt neue Heransgehensweisen für seine Analyse. Im nächsten Abschnitt stellen wir Techniken zur Analyse der Bedienbarkeit vor.

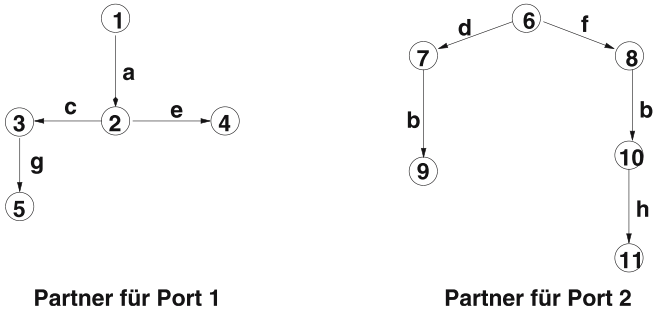

Abb. 4 Ein mögliches Paar von Partnern des oWFN aus Abb. 3. Sie bedienen das oWFN. Würde sich Partner 1 allerdings wie Partner 2 verhalten (also durch den gleichen Automaten, nur mit jeweils den Vorgängerbuchstaben beschriftet), so würde das System in dem Zustand verklemmen, wo alle Nachplätze von $t1$ oder die von $t2$ markiert sind

4 Analyse der Bedienbarkeit

Bedienbarkeit verlangt die *Existenz* bestimmter Partner für ein gegebenes oWFN. Wir stellen nun Entscheidungsverfahren für die Bedienbarkeit *azyklischer* oWFN vor, also von oWFN, in denen keine Markierung von sich selbst wieder erreichbar ist.

4.1 Automaten als Partner

Unser Verfahren ist konstruktiv, im Fall der Bedienbarkeit wird also ein Satz bedienender Partner explizit angegeben. Für die Berechnung der Partner gehen wir jedoch einen Umweg. Statt direkt oWFN zu berechnen, konstruieren wir zunächst Automaten, die die gleiche Rolle wie die gesuchten Partner spielen. Aus diesen Automaten berechnen wir dann oWFN derart, dass ihr Erreichbarkeitsgraph genau den konstruierten Automaten entspricht. Für die Berechnung eines oWFN aus einem Automaten verweisen wir auf die Techniken aus [3, 6, 9, 19] und gehen hier nicht weiter darauf ein.

Wir betrachten im Rest dieses Abschnittes also klassische endliche Automaten. Da sie letztlich das Verhalten von Partnern beschreiben, nennen wir sie ebenfalls *Partner*. Abbildung 4 zeigt zwei solche Automaten. Einen Partner, der den Port π benutzt, modellieren wir als Automat, der auf dem Alphabet π operiert. Einen Übergang mit einem Ausgangskanal des Netzes bezeichnen wir demnach als *Empfangsaktion* und einen Übergang mit einem Eingangskanal des Netzes als *Sendeaktion* des Partners.

Das Zusammenspiel eines oWFN N mit einer Menge $\mathcal{M}$ von Partnern beschreiben wir als Transitionssystem, das *Kooperationssystem* von N mit $\mathcal{M}$. Abbildung 5 zeigt das Kooperationssystem des oWFN aus Abb. 3 mit den beiden Partnern in Abb. 4.

Ein Zustand eines Kooperationssystems ist ein Tupel, das aus einer Markierung von N sowie je einem Zustand der Partner besteht. Jeder Übergang eines Partnerautomaten und jedes Schalten einer Transition des oWFN definiert je einen Übergang im Kooperationssystem wie folgt. Ein mit x beschrifteter Übergang eines Partners bewirkt, dass eine Marke

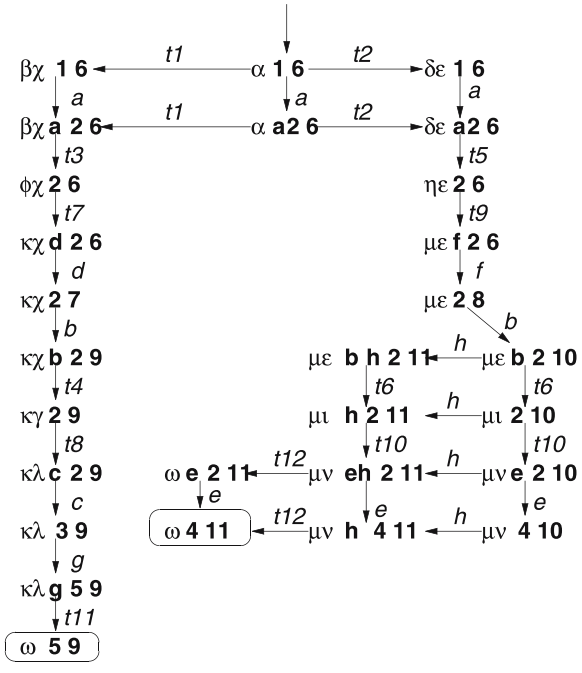

Abb. 5 Kooperationssystem für das oWFN aus Abb. 3 und die Partner aus Abb. 4. Jeder Übergang im Kooperationssystem entspricht einem Übergang im oWFN oder einem der Partner. Dabei bewirkt ein Übergang eines Partners die Änderung der Markierung eines Schnittstellenplatzes des oWFN

auf Schnittstellenplatz x von N erzeugt bzw. entfernt wird. Wir verzichten hier auf eine formale Definition, merken aber an, das diese Definition genau den Kompositionsbegriff für oWFN nachbildet.

Wir unterscheiden für das Bedienbarkeitsproblem drei Fragestellungen. Die erste Fragestellung nennen wir *zentrale Bedienbarkeit*. Hier untersuchen wir oWFN, die genau einen Port besitzen.

Im zweiten Fall, der *verteilten Bedienbarkeit*, betrachten wir oWFN mit mehr als einem Port.

Der dritte Fall, *autonome Bedienbarkeit*, ist ein Spezialfall verteilter Bedienbarkeit. Hier fragen wir, ob sich die

Partner für die einzelnen Ports sogar unabhängig voneinander konstruieren lassen.

Für alle drei Fragestellungen bieten wir Lösungen für den Fall an, dass das oWFN azyklisch ist. Sämtliche in diesem Abschnitt gezeigten Beispielnetze sind azyklisch.

4.2 Entscheidungsverfahren für zentrale Bedienbarkeit

Zur Lösung des zentralen Bedienbarkeitsproblems ist es unser Ziel, einen bedienenden Partner für ein gegebenes oWFN N zu konstruieren, wann immer einer existiert. Ein solcher Partner muss mit N geeignet kommunizieren. Ob eine bestimmte Nachricht in einer bestimmten Situation geeignet oder ungeeignet ist, hängt natürlich davon ab, in welcher Markierung sich N gerade befindet. Die Entscheidung, ob ein Partner eine Nachricht sendet oder auf den Empfang einer Nachricht wartet, muss der Partner aber treffen, ohne direkten Zugriff auf diese Markierung zu haben. Zur Entscheidung kann er lediglich die jeweils aufgelaufene Kommunikation heranziehen. In Kenntnis von N kann man aus der aufgelaufenen Kommunikation Rückschlüsse auf die Markierung von N ziehen. Man kann zwar im allgemeinen nicht exakt bestimmen, in welcher Markierung sich N befindet, kann jedoch die Menge der möglichen Markierungen eingrenzen. Das „Wissen“ eines Partners über bisherige Kommunikation spiegelt der jeweils erreichte Zustand des Partners wieder. Die Rückschlüsse aus der aufgelaufenen Kommunikation auf die Markierungen, in denen sich N befinden kann, können wir also darstellen als eine Abbildung K (*knowledge*), die jedem Automatenzustand q des Partners die Menge derjenigen Markierungen zuordnet, in denen sich N befinden *kann*, während der Partner sich in q befindet. Zu gegebenem N und einem beliebigen gegebenen Partner können die Werte $K(q)$ aus dem zugehörigen Kooperationssystem abgeleitet werden.

Mit Hilfe dieser Zuordnung können wir ein Kriterium für bedienende Partner aufstellen.

Theorem 1 (Charakterisierung bedienender Partner, [23]) *Ein einzelner Partner M bedient ein oWFN N genau dann, wenn für alle Zustände q von M und alle Markierungen $m \in K(q)$ gilt: Wenn bei m keine Transition von N schalten kann, so ist m eine Endmarkierung oder es gibt in q Übergänge für M, die in $[m, q]$ im Kooperationssystem aktiviert sind.*

Dieses Kriterium liefert ein Verfahren zur Konstruktion eines bedienenden Partners: Wir konstruieren den Partner aus einem Automaten, der in jedem Zustand jede mögliche Sende- und Empfangsaktion vorsieht, also eher mehr als das letztlich akzeptable Verhalten umfasst. Diesen Automaten können wir als Baum konstruieren, dessen Tiefe der erwähnten Obergrenze für Kommunikationsschritte entspricht, und in dem von jedem inneren Knoten mit jedem Zeichen des Alphabets (also mit jedem Schnittstellenplatz des gegebenen oWFN) ein Übergang vorhanden ist. Zu jedem Zustand q dieses Automaten kann man nun den Wert $K(q)$ berechnen

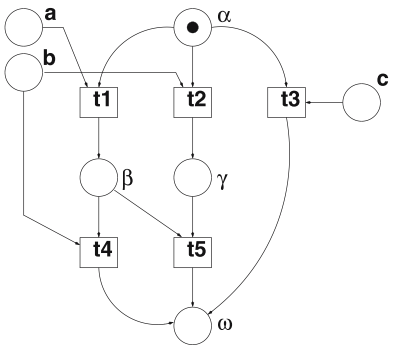

Abb. 6 Ein oWFN

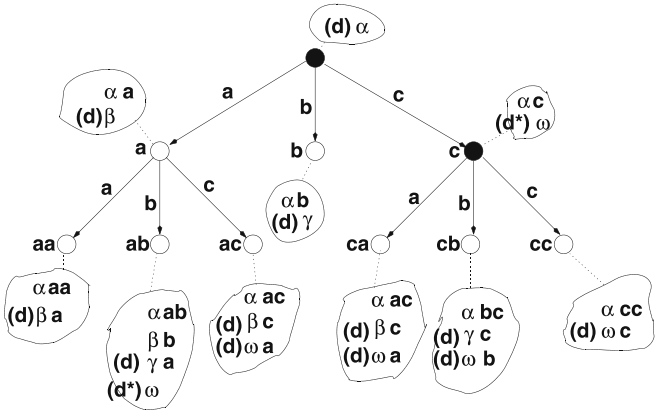

Abb. 7 Konstruktion eines bedienenden Partners für das oWFN N in Abb. 6. Gezeigt ist der Automat, mit dem man die Konstruktion bedienender Partner beginnt, zusammmen mit den Werten der Wissensfunktion K, wobei Markierungen, bei denen in N keine Transitionen aktiviert sind, mit **(d)** gekennzeichnet sind. Die Zustände sind mit Sequenzen aus Schnittstellenplätzen beschriftet, zeigen also die bis dahin stattgefundene Kommunikation. Wir streichen zunächst alle *terminalen* Knoten, wo ein mit **(d)** gekennzeichneter Zustand möglich ist, der keine Endmarkierung **(d*)** ist. Nach dem Streichen des Zustands links unten muss auch dessen Vorgänger gestrichen werden, weil nun auch er keinen Nachfolger hat, aber einen mit **(d)** gekennzeichneten Zustand in $K(q)$ enthält. Es verbleibt der Automat, der aus den ausgefüllt dargestellten Zuständen besteht. Dieser bedient das Netz in Abb. 6

und in $K(q)$ Markierungen bestimmen, in denen keine Transition von N aktiviert ist. Solange es Zustände gibt, die das Kriterium aus Thm. 1 verletzen, werden diese Zustände sowie die bei diesen Zuständen beginnenden Teilbäume gestrichen. Wir verzichten hier auf die Präsentation von Pseudocode und verweisen stattdessen auf das in Abb. 7 dargestellte Beispiel der Berechnung eines bedienenden Partners für das oWFN in Abb. 6.

Terminiert die Konstruktion mit einem nichttrivialen Automaten (also einer nichtleeren Zustandsmenge), ist das Netz bedienbar. Führt die Konstruktion dagegen zu einem Automaten mit leerer Zustandsmenge, ist das Netz nicht bedienbar.

Diese Konstruktion ist in dieser Form für große Netze sicher nicht durchführbar, weil der Automat, mit dem wir starten, zu viele Zustände hat. Um diesem Problem zu begegnen, haben wir bereits eine Reihe von Reduktionen vorgeschlagen [27], die genau diese Zustandsexplosion bekämpfen.

Auf der Basis unserer Konstruktion können wir zeigen, dass es zu jedem oWFN einen eindeutig bestimmten *liberalsten* bedienenden Partner gibt, also einen, aus dem einzig durch weitere Verhaltenseinschränkung jeder beliebige andere bedienende Partner abgeleitet werden kann. Mit Hilfe des liberalsten Partners kann leicht *vollständige* Bedienbarkeit untersucht werden: Ein oWFN ist vollständig bedienbar genau dann, wenn jede Transition bei Komposition mit dem liberalsten bedienenden Partner aktivierbar ist.

4.3 Entscheidungsverfahren für Verteilte Bedienbarkeit

Wir stellen nun ein Entscheidungsverfahren für den Fall vor, dass ein oWFN mehrere Ports hat. Beispielsweise bedienen die beiden Partner aus Abb. 4 zusammen das oWFN in Abb. 3 verteilt.

Wenn ein oWFN verteilt bedienbar ist, kann man die bedienenden Partner zusammen als *einen* Partner auffassen, der das oWFN zentral bedient (nach Vereinigung aller Ports zu einem einzigen): Mit einem geeigneten Begriff für parallele Komposition ist der zentrale Partner die parallele Komposition der verteilt bedienenden Partner.

Umgekehrt kann man die Frage nach verteilt bedienenden Partnern auffassen als die Frage nach einem zentral bedienenden Partner, der die Form einer parallelen Komposition verteilter Partner hat. Diese Form hat ein zentraler Partner, falls die zu verschiedenen Ports gehörenden Übergänge sich nicht gegenseitig beeinflussen. Ein Übergang von q nach q' mit x *beeinflusst* einen Übergang von q_1 nach q_1' mit y, wenn im zentral bedienenden Partner eine der folgenden Bedingungen gilt:

- x aktiviert y: $q' = q_1$ und es gibt keinen Übergang mit y bei q;
- x deaktiviert y: $q = q_1$ und es gibt keinen Übergang mit y bei q';
- Die Reihenfolge von x und y ist relevant: Es gibt einen Übergang von q' zu einem q'' mit y und einen Übergang von q_1' zu einem q_2'' mit y, die bei q'' und q_1'' beginnenden Teilbäume des Partners sind aber nicht isomorph.

Theorem 2 (Verteilte Bedienbarkeit [23]) *Ein oWFN ist verteilt bedienbar genau dann, wenn es einen zentral bedienenden Partner gibt, in dem sich die zu verschiedenen Ports gehörenden Übergänge nicht gegenseitig beeinflussen.*

Eine Implementierung der gezeigten Idee führt zu einem Algorithmus, der mit Backtracking arbeitet. Wir verzichten hier auf eine Darstellung dieses Algorithmus. Backtracking als Bestandteil einer Implementation scheint unumgänglich, weil wir es, wie Abb. 8 zeigt, mit *Symmetriebruchproblemen*[3] zu tun haben, für die Backtracking eine der wenigen Lösungsmöglichkeiten darstellt. Das Beispiel zeigt weiterhin, dass es im verteilten Fall nicht notwendigerweise eine eindeutig bestimmte „liberalste" Menge bedienender Partner gibt.

[3] Das Problem, in einem System gleichartiger Partner aus einem symmetrischen in einen unsymmetrischen Zustand überzugehen.

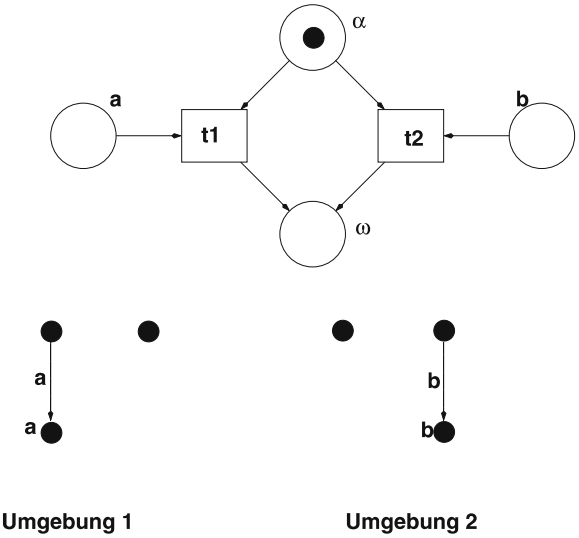

Abb. 8 Für das oben abgebildete oWFN lassen sich die beiden darunter dargestellten bedienenden Partnerpaare berechnen. Beide Partner zusammen bedienen das oWFN jeweils verteilt. Obwohl das Netz selbst symmetrisch ist, sind die beiden bedienenden Umgebungen nicht in sich symmetrisch (sondern nur zueinander). Es gibt für das oWFN kein symmetrisches Paar bedienender Partner

4.4 Autonome Bedienbarkeit

Für das Problem der verteilten Bedienbarkeit wird eine Menge von Partnern konstruiert, die gemeinsam das Netz bedienen. Der Konstrukteur der Partner kann dabei jeden Partner in Kenntnis der anderen Partner konstruieren. In Abb. 8 kann er beispielsweise den Partner für Port $\{b\}$ nur dann so bilden, dass dieser gar keine Aktion ausführt, wenn der Partner für Port $\{a\}$ tatsächlich die Aktion a ausführt.

Hier fragen wir uns nun, inwieweit es sinnvoll und überhaupt möglich ist, einen einzelnen der beteiligten Partner *ohne* Kenntnis der anderen Partner zu konstruieren. Wir suchen also einen Partner, der mit *beliebigen* anderen Partnern zusammen das gegebene oWFN bedient. In dieser Fassung ist die Fragestellung allerdings trivial: Beliebig „destruktive" andere Partner können Bedienbarkeit immer verhindern. Wir beschränken uns deshalb auf *kooperative* Partner und definieren zunächst diesen Begriff.

Im weiteren sei N ein oWFN mit den Ports $\pi_1, \ldots, \pi_k$ und π_i ein ausgezeichneter Port. Im weiteren definieren wir, wann ein Partner eine Markierung *beachtet*, bei der keine Transition von N aktiviert ist, wie das auf π_i *reduzierte Netz* N_{π_i} aussieht, wann ein Partner von N *kooperativ* ist und schließlich, wann ein oWFN *autonom bedienbar* heißt.

Sei U ein Partner, der Port π_i benutzt, also, wie in Abschn. 3.3 definiert, ein endlicher Automat mit Alphabet π_i. Für einen Zustand q von U kann die in Abschn. 4.2 definierte Menge $K(q)$ Markierungen enthalten, in denen keine Transition von N aktiviert ist. Sei m eine solche Markierung, z.B. die Markierung $\beta\chi$ des Netzes in Abb. 3. Für m betrachten wir die Menge E derjenigen Eingangskanäle, die in m unmarkiert sind, deren Markierung aber wenigstens eine

Transition von N aktivieren würde. Für die Markierung $\beta\chi$ wäre $E = \{a, b\}$. *U beachtet m*, falls kein Eingangskanal aus E zum Port von U gehört oder es in U einen Übergang bei q gibt, der in *m möglich* ist, also entweder eine beliebige Sendeaktion oder eine Empfangsaktion von einer bei m markierten Schnittstelle.

Es mag verwundern, dass wir in *q nicht* unbedingt eine Sendeaktion auf ein Element aus E fordern. Diese Liberalität ist einerseits notwendig, weil U in q Gelegenheit haben soll, vielleicht noch einige Empfangsaktionen auszuführen, oder Sendeaktionen, die später relevant sind, bereits vorzuziehen. Andererseits ist die Liberalität auch ausreichend, weil gegebenenfalls die bei m deaktivierten Transitionen deaktiviert bleiben und das Problem also nur aufgeschoben wird. Unendlichen Aufschub können wir durch eine Abschätzung der Höchstzahl der Kommunikationsschritte mit dem (azyklischen!) oWFN ausschließen.

Das oWFN N_{π_i} reduziert N auf die Schnittstellenplätze des Ports π_i. N_{π_i} entsteht aus N durch Streichen aller Schnittstellen, die nicht in π_i liegen, sowie durch Streichen der angrenzenden Kanten. Der Partner U ist *kooperativ*, wenn

- er jede Markierung von N beachtet, in der keine Transition von N aktiviert ist und bei der wenigstens ein unmarkierter Eingangskanal in π_i liegt, und
- N_{π_i} zentral bedient.

Zum oWFN in Abb. 3 ist Partner 1 aus Abb. 4 kooperativ, Partner 2 dagegen nicht. Die Markierung $\beta\chi$ liegt in $K(6)$ (dem Anfangszustand von Partner 2) und der von Partner 2 benutzte Eingangskanal b ist unmarkiert. Partner 2 beachtet diese Markierung nicht, weil er im Zustand 6 lediglich die in $\beta\chi$ nicht möglichen Empfangsaktionen d und f als Übergänge hat.

Ein oWFN nennen wir *autonom bedienbar*, falls es zu jedem seiner Ports einen *kooperativen* Partner gibt.

Kooperative Partner sind deshalb interessant, weil kooperative Partner in der Lage sind, mit jedem beliebigen anderen kooperativen Partner zusammenzuarbeiten:

Theorem 3 ([23]) *Jede Partnermenge, die für jeden Port eines oWFN einen kooperativen Partner enthält, bedient das oWFN.*

Die Existenz kooperativer Partner (und also autonome Bedienbarkeit) können wir, für jeden Port einzeln, mit einem Algorithmus entscheiden, der mit dem für zentrale Bedienbarkeit fast identisch ist.

Verhalten sich beide Partner aus Abb. 4 nach dem Schema wie Partner 1 (also kooperativ), terminiert das oWFN in Abb. 3, während es in den Markierungen $\beta\chi$ bzw. $\delta\epsilon$ verklemmen kann, wenn beide sich wie Partner 2 verhalten. Für das oWFN in Abb. 8 existieren keine kooperativen Partner; es ist nicht autonom bedienbar.

5 Zusammenfassung

Neben der Bedienbarkeit gibt es eine Reihe weiterer wichtiger Fragestellungen an Modelle kommunizierender Workflow-Services, beispielsweise die Konstruktion eines *public views* oder von *Bedienungsanleitungen*, Äquivalenzbegriffe und Probleme semantikerhaltender Verfeinerung, sowie das weite Feld transaktionalen Verhaltens. Einige Fragestellungen betreffen nur die Kontrollstruktur kommunizierender Workflow-Services, andere sind auch von Datenaspekten abhängig (beispielsweise die Konstruktionen zur Semantik von BPEL in Abschn. 2). Petrinetze bieten Varianten an, die genau auf beide Aspekte zugeschnitten sind und in engem Zusammenhang stehen.

Zusammengefasst formuliert, sollen die in diesem Beitrag dargestellten Konzepte und Algorithmen zeigen, dass die oben geschilderten Begriffe mit Petrinetzen angemessen modelliert und computergestützt analysiert werden können.

Literatur

1. Aalst W (1998) The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers 8(1):21–66
2. Aalst W, Hofstede A, Kiepuszewski B, Barros AP (2003) Workflow patterns. Distrib Parallel Databases 14(1):5–51
3. Badouel E, Darondeau P (1998) Theory of regions. LNCS: Lectures on Petri Nets I: Basic Models 1491:529–586
4. Casati F, Ceri S, Pernici B, Pozzi G (1995) Conceptual Modeling of Workflows. In: Papazoglou MP (ed) Proceedings of the OOER'95, 14th International Object-Oriented and Entity-Relationship Modelling Conference, Springer-Verlag, vol 1021, pp 341–354
5. Clarke E, Grumberg O, Jha S, Lu Y, Veith H (2003) Counterexample-guided abstraction refinement for symbolic model checking. J ACM 50(5):752–794
6. Cortadella J, Kishinevsky M, Lavagno L, Yakovlev A (1998) Deriving petri nets from finite transition systems. IEEE Trans Comput 47(8):859–882
7. Curbera F, Goland Y, Klein J, Leymann F, Roller D, Weerawarana S (2003) Business Process Execution Language for Web Services, Version 1.1. Tech. rep., BEA Systems, International Business Machines Corporation, Microsoft Corporation
8. Dehnert J, Rittgen P (2001) Relaxed soundness of business processes. LNCS: Advanced Information System Engineering, CAISE 2001 2068:157–170
9. Desel J, Reisig W (1996) The Synthesis Problem of Petri Nets. Acta Informatica 33(4):297–315
10. Fahland D (2005) Complete Abstract Operational Semantics for the Web Service Business Process Execution Language. Tech. Rep. 190, Humboldt-Universität zu Berlin
11. Farahbod R, Glässer U, Vajihollahi M (2004) Specification and Validation of the Business Process Execution Language for Web Services. In: Abstract State Machines, Springer, LNCS, vol 3052, pp 78–94
12. Ferrara A (2004) Web services: a process algebra approach. In: ICSOC, ACM, pp 242–251
13. Fisteus JA, Fernández LS, Kloos CD (2004) Formal Verification of BPEL4WS Business Collaborations. In: Proceedings of the 5th International Conference on Electronic Commerce and Web Technologies (EC-Web '04), Springer, LNCS
14. Fu X, Bultan T, Su J (2004) Analysis of interacting BPEL web services. In: WWW '04: Proceedings of the 13th international conference on World Wide Web, ACM Press, pp 621–630
15. Hinz S (2005) Implementation einer Petrinetz-Semantik für BPEL4WS. Diplomarbeit, Humboldt-Universität zu Berlin
16. Hinz S, Schmidt K, Stahl C (2005) Transforming BPEL to Petri Nets. In: accepted for Third International Conference on Business Process Management (BPM 2005), Nancy, France, September 2005
17. Hollingsworth D (1995) The workflow reference model. Tech. Rep. TC00-1003, Workflow Management Coalition

18. Foster H, Magee J, Uchitel S, Kramer J (2003) Model-based Verification of Web Service Compositions. In: 18th IEEE International Conference on Automated Software Engineering, IEEE Computer Society, pp 152–163
19. Nielsen M, Rozenberg G, Thiagarajan PS (1992) Elementary transition systems. Theoretical Computer Science 96
20. Oberweis A (1996) Modellierung und Ausführung von Workflows mit Petri-Netzen. Teubner Reihe Wirtschaftsinformatik, B.G. Teubner Verlag, Stuttgart, Germany
21. Schmelzer HJ, Sesselmann W (2004) Geschäftsprozessmanagement in der Praxis, 4th edn. Hanser
22. Schmidt K (2000) Lola – a low level analyser. In: Nielsen M, Simpson D (eds) International Conference on Application and Theory of Petri Nets, Springer-Verlag, LNCS, vol 1825, p 465 ff
23. Schmidt K (2004) Controlability of Business Processes. Tech. Rep. 180, Humboldt-Universität zu Berlin
24. Schmidt K, Stahl C (2004) A Petri net semantic for BPEL4WS – validation and application. In: Kindler E (ed) Proceedings of the 11th Workshop on Algorithms and Tools for Petri Nets (AWPN'04), Universität Paderborn, pp 1–6
25. Stahl C (2004) Transformation von BPEL4WS in Petrinetze. Diplomarbeit, Humboldt-Universität zu Berlin
26. Stahl C (June, 2005) A Petri Net Semantic for BPEL. Tech. Rep. 188, Humboldt-Universität zu Berlin
27. Weinberg D (2004) Analyse der Bedienbarkeit. Diplomarbeit, Humboldt-Universität zu Berlin
28. White SA (2004) Business Process Modelling Notation Version 1.0. Tech. rep., Business Process Management Initiative (BPMI)

Wolfgang Reisig studierte Mathematik in Karsruhe und in Bonn. Er arbeitete als Assistent an der RWTH Aachen, wo er 1979 promovierte. Von 1983–1987 arbeitete er als Projektleiter bei Dr. Petri. Zwischen 1987–1993 war er als Professor an der TU München tätig. Seither leitet er den Lehrstuhl „Theorie der Programmierung" an der Humboldt-Universität zu Berlin. Sein Forschungsschwerpunkt sind formale Modelle für verteilte und reaktive Syteme.

Karsten Schmidt promovierte und habilitierte an der Humboldt-Universität zu Berlin. Er arbeitete außerdem bereits an der Helsinki University of Technology, der Technischen Universität Dresden und der Carnegie Mellon University in Pittsburgh. Er arbeitet als wissenschaftlicher Mitarbeiter am Lehrstuhl „Theorie der Programmierung" an der Humboldt-Universität hauptsächlich im Gebiet Model Checking sowie der Modellierung und Analyse von Workflows.

Christian Stahl studierte Informatik an der Humboldt-Universität zu Berlin. Er arbeitet als wissenschaftlicher Mitarbeiter am Lehrstuhl „Theorie der Programmierung" an der Humboldt-Universität. Seine Forschungsschwerpunkte sind Modellierung und Verifikation verteilter Systeme, vor allem Workflows und asynchrone Hardware.